

4 Design Standards

This section describes the standards for objects that will be encountered or defined in the Design phase of application development.

4.1 Database Design Transformer

4.1.1 Settings

4.1.1.1 Database

DES-01 The database name selected is “<none>”. This means that the database and tablespace settings in Designer may be left blank. The table definitions are initially generated without defining/implementing against a database, tablespace or storage clause. This is also true for the generation of the index definition under the table definition.

4.1.1.2 Keys

DES-02 The cascade rules for the newly generated foreign key definitions should be the default value of “RESTRICTED”. Cascade rules will depend on business rules and should be handled on a case by case basis if the default is not used. Document any deviations from using RESTRICTED in the entity model description.

DES-03 Surrogate keys are to be defined using the shared sequence domain appropriate for the sequence. Surrogate keys are named using the convention:

- SEQ_ID
- SEQ_V n

DES-04 The maximum identifier length should be the default value of 30.

4.1.1.3 Other Settings

DES-05 The ordering of the Columns should be as follows:

- Primary Key Columns (major to minor)
- Mandatory Columns (including User Audit Columns)
- Discriminator Columns
- Foreign Key Columns before Attribute Columns
- Grouping Columns by their Source Entity
- LONG Datatypes

NOTE: Long and long raw columns should be moved to the bottom of the column list after the table is generated. Only one of these types of columns is allowed.

4.2 Tables

4.2.1 Naming Convention

DES-06 Define table names in singular. Name tables derived from an entity using the convention:

<application prefix>_<entity_name_in_singular>

Refer to Appendix B for the application prefix names.

DES-07 Tables that implement an entity must have the same name as the entity they implement, with the spaces translated to underscores. If the table is not based on an entity, then it must be named using the DFAS standard abbreviations, using underscores between segments.

Table names must be unique within the first 19 characters. This allows for the creating of snapshots when a two-character prefix is added to the table name. If the resulting name has more than 26 characters, work with DFAS Data Administration to shorten the table name.

4.2.2 Definition

DES-08 The Designer created short name alias should be kept to 3 characters whenever possible to prevent names from becoming too long. The maximum alias length is 6 characters.

DES-09 Enter the full name of the table/view in the Display Title field for each table/view. These names will be seen by end users. The full name is found in the “Long Name” property.

DES-10 Do not deviate from the default value for Init Trans, unless you predict that the table will form a “hot spot”, requiring many users to update a small number of rows simultaneously. Document any deviation from the default value in the Description belonging to the table definition, using the keyword INIT TRANS.

DES-11 Do not specify values for both PCTFREE and PCTUSED. For large tables, specify in the Description what the source of this information is, using the keywords PCTFREE/ PCTUSED; include the sample size. (Base the values of PCTFREE and PCTUSED on the guidelines presented in the ORACLE8 Server Administrator’s Guide.)

DES-12 Expand the description of a table to include any design level requirements. The initial description came from the entity description.

DES-13 Define the purpose and usage in detail for tables that do not directly implement an entity.

DES-14 All table definitions will be generated from the appropriate entity or entities in the logical data model and will not be created manually. This does not apply to tables which are created specifically to facilitate the physical implementation of a function.

DES-15 All tables must have an alias. The alias must conform to the same standard set forth for entity short names. If the table is based on an entity, then the alias must be the same as the entity short name.

DES-16 To ensure that the appropriate meta-data is included in the Oracle data dictionary, all tables must have a comment entered into Designer. This comment should describe the basic information stored in the table.

4.3 Columns

4.3.1 Naming Convention

- DES-17 Define column names in singular.
- DES-18 Do not use the table alias as a prefix in column names with the exception of foreign key columns.
- DES-19 If the resulting name has more than 30 characters, use the approved acronyms and abbreviations in the appendices to shorten it.
- DES-20 Do not start column names with P_. The Forms Generator confuses such a column with parameters and will fail with an error message.
- DES-21 Columns that implement an attribute should have the same name as the attribute they implement, with the spaces translated to underscores. If the column is not based on an attribute, then it should be named using the naming standard set forth for attributes with the exception that an underscore is used instead of a space between segments.
- DES-22 Keep column names short, but still logical and self-descriptive. Do not repeat the table name in the column name. Word abbreviations in the column name must follow the approved DFAS abbreviation conventions. (Appendix B)
- DES-23 If you choose to resolve a sub-type design using one table, introduce a discriminator column to distinguish sub-types using the naming convention:
- `<super_entity_shortname>_TY`
- DES-24 Define the discriminator column with datatype VARCHAR2(6), with the sub-type entity short names as allowable values.
- DES-25 Define system-generated primary key columns using the name ID.
- DES-26 Name foreign key columns using the convention:
- `<table_alias_referenced_table>_<primary_key_column_name_referenced_table>`
- OR*
- For multiple foreign keys to the same table:
- `<table_alias_referenced_table>_<relationship identifier>`

4.3.2 Definition

- DES-27 Assign to columns in a view the same name as the columns in the underlying tables. For columns not directly based on database columns, refer to the conventions for naming columns. For columns from different tables, but with the same column name, prefix the column name with the table alias even if only one of the columns is used in the view.
- DES-28 Any column that ranges over a fixed set of predefined values that is less than 21 values should be associated with a static domain that describes that set of values.

- DES-29 Any column that ranges over a dynamic set of values that is less than 21 values should be associated with a dynamic domain that describes that set of values.
- DES-30 Any column that ranges over 20 values should be placed in a reference table.
- DES-31 Define indicator columns that are used to select a small set in a table as optional and VARCHAR2(1) with allowable values Y and N. If the options are 3-way logic, describe why this is necessary.
- DES-32 For columns of datatype RAW and LONG RAW, specify in the column Description the internal format that is stored in the column by referencing the industry standard (if any). Use the keyword DATATYPE RAW in the Description.
- DES-33 Columns that are optional should have a short column note that explains the meaning of a null value occurring for that column, if the meaning is different from VALUE UNKNOWN.
- DES-34 Define discriminator columns indicating a sub-type as NOT NULL.
- DES-35 Define the initial volume for each column (100% for mandatory columns). Specify the source of your estimate for optional columns, if any, in the column Description using the keyword VOLUME.
- DES-36 Define the final volume for each column (100% for mandatory columns). Specify the source of your estimate for optional columns, if any, in the column Description using the keyword VOLUME.
- DES-38 When a check box is defined as the display type, the normal default value must be set to a value within the domain. Leave this field blank only when the check box column is optional, which implies that the domain has only one allowable value and the other value is NULL. The check box for an optional column with two allowable values will always insert one of those values and the column will only be NULL at initialization of the record.
- DES-39 If a column is defined in a view, and the underlying table column has a sequence associated with it, you should define the same sequence here.
- DES-40 All referential and transactional tables require the standard audit columns. These are:
- | COLUMN NAME | DATA TYPE | DOMAIN |
|---------------|--------------|--------|
| CREATED_BY | VARCHAR2(30) | TXT030 |
| DATE_CREATED | DATE | |
| MODIFIED_BY | VARCHAR2(30) | TXT030 |
| DATE_MODIFIED | DATE | |
- DES-41 Do not deviate from the standard database column datatype for non-displayed columns.
- DES-42 Use a check box when only one value is applicable in a yes/no situation, and the yes/no statement is not contrived or obscure. A check box can be initialized to NULL, but it cannot be set to NULL.
- DES-43 Use a Boolean set when a maximum of one of out of two values is applicable, and if the list will be static throughout the life of the product. A Boolean set can be set to NULL.
- DES-44 Use radio group, radio group (meaning) or radio group (abbreviation) when a maximum of one of two to five values is applicable, and if the list will be static throughout the life of the product.
- DES-45 Use pop list, pop list (meaning) or pop list (abbreviation) when only one of three to fifteen values is applicable, and the list is never expected to grow beyond fifteen.

- DES-46 Use LOV window when only one of five to twenty values is applicable, and the list is dynamic during the life of the product. (All dynamic domains and reference tables should be displayed using a LOV window.
- DES-47 Avoid text list, text list (meaning) and text list (abbreviation) due to the amount of space they require. If using text lists, use them for lists of between three and twenty entries.
- DES-48 Use combo box, combo box (meaning) or combo box (abbreviation) if all of the following is true:
- You have a list of allowable values that will be used most of the time.
 - The user already knows that this list does not cover all situations.
 - While at the same time the user is not able to complete the list.
- DES-50 Foreign key columns should be displayed in the same sequence as their primary key counterparts.
- DES-51 Do not specify the highlighting option at the column level, but rather at the module level.
- DES-52 The first letter of any word in a prompt is capitalized. Prompts should clearly indicate to what property the column refers.
- DES-53 Hint texts take the form of the remainder of the sentence, “The value in this field registers <hint text>”.
- DES-54 Hint texts for indicator columns use a question form. Do not include the allowed values in the hint text, since this might complicate the repopulating of domain values.
- DES-55 Define a descriptor column in all situations where the primary key has no meaning of its own.
- DES-56 Only use the type “Seq In Parent” (Sequence within Parent) in addition to server-side implementation when this field needs to be displayed in the generated form.
- DES-57 If the column has a Derivation Expression that implements a Tuple Rule, you should record a reference to this rule here.
- DES-58 Set the Suggestion List property to ‘checked’ if you use Combo Box as display datatype for the column.
- DES-59 Only use Value to enter a lower limit for a column.
- DES-60 Only use High Value to enter a higher limit for a column.
- DES-61 Do not use abbreviations. Use the domain object if you want to record an enumerated domain.
- DES-62 Do not use meanings. Use the domain object if you want to record an enumerated domain.
- DES-63 The nature of a derivation imposes implementation at the server-side using a database trigger. Use this expression only if the value of the derived column must be displayed on the screen. You should still implement the derivation expression at the server-side, ensuring that the derived value is only calculated if not yet done at the client-side.
- DES-64 The Derivation Expression of the column must not refer to itself. The derived column also must not be defined to derive its value from an internal source, such as a generated sequence number, a username, or sysdate. The column should not have a default value.
- DES-65 Do not use the Where/Validation Expression on the column level. Instead, define any Where/Validation Expression on the table level in the Table Key Constraints Definition tabsheet.

DES-66 Describe the purpose of the column if this is not clear from the column name itself.

4.4 Constraints

4.4.1 Naming Convention

DES-67 Constraint names will conform to the default Designer constraint naming conventions that are automatically created when using the Database Design Transformer. Any manually created constraints must conform to this standard.

4.4.1.1 Primary Key Constraint

DES-68 Name primary key constraints using the convention::

<source_table/view_alias>_PK

Example:

For the table CUSTOMER with the alias CUST the primary key will be: CUST_PK

4.4.1.2 Unique Key Constraints

DES-69 Name Unique Key Constraints using the convention:

<table alias>_<UID Name>_UK

Where the UID Name is the name specified for a secondary unique identifier on the entity.

Example: The table called ORDERS (ORD) has two unique identifiers named ORD2 and ORD3. The unique key constraints will be named:

ORD_ORD2_UK

ORD_ORD3_UK

4.4.1.3 Foreign Key Constraints

DES-70 Name foreign key constraints using the convention:

When there is only one foreign key constraint:

<source_table/view_alias>_<ref_table/view_alias>_FK

When there are multiple foreign key constraints to the referenced table, the relationship identifier should provide a meaningful name to identify the purpose of the foreign key constraint:

<source_table/view_alias>_<relationship_identifier>_FK

Designer will generate one of two possible names depending on the number of relationships between two entities. Foreign key names are usually generated with the following algorithm:

Foreign key = <table alias of the target table>_<table alias of the originating key>_FK

Example:

ORDER_LINES (ORDLIN) >----- ORDERS (ORD)

The foreign key constraint on ORDER_LINES will have the foreign key constraint name generated as ORDLIN_ORD_FK.

However, if a table has multiple foreign keys to one other single table, then the algorithm for the first foreign key constraint will be the same as above. However, the second constraint will be generated as follows:

Foreign key = <table alias of the target table>_<table alias of the originating key>_relationship name_FK

Where 'relationship name' is the text associated with the relationship as depicted on the logical model.

Example:

```
ORDERS >----- LOCATIONS (LOC) shipped from
        (ORD) >----- LOCATIONS returned to
```

The foreign key constraints on ORDERS will have the foreign key constraint names generated as:

```
ORD_LOC_FK
ORD_LOC_RETURNED_TO_FK
```

For the sake of clarity, in this case the first FK constraint should be renamed to:

```
ORD_LOC_SHIPPED_FROM_FK
```

4.4.1.4 Check Constraints

DES-71 Name check constraints using the convention:

<table/view_alias><optional_number><_><optional constraint column name>_CK

Both optional values are to be used as desired by the designer.

4.4.1.5 Columns

DES-71.1 Columns which participate in unique or primary keys should be upper case. If data or business rules require mixed or lower case, add the string "Mixed case required" to the notes property for that column.

4.4.2 Definition

4.4.2.1 Primary Key Constraint

DES-72 Set the "Validate In" property to BOTH for primary keys of a table. If for any reason you deviate from this standard, document the reasons in the Primary Key Description.

DES-73 Set the "Validate In" property to CLIENT for primary keys of a view. If for any reason you deviate from this standard, document the reasons in the Constraint Description.

DES-74 Define all columns that are part of the primary key as not updateable. If any of the columns in the primary key need to be updated, make the current primary key an alternate key and introduce a system-generated key to serve as primary key.

DES-75 Define all columns that are part of the primary key as not null.

DES-76 For tables, document the cause of the Primary Key constraint not being enabled.

4.4.2.2 Unique Key Constraints

DES-77 Columns in a unique key constraint should either be all defined as NULL or all defined as NOT NULL. Whenever mixed NULL and NOT NULL columns are used, the reason should be documented in the notes section of the unique key constraint.

If the columns in a unique key constraint are defined as NULL, your application code should enforce that for each row in the table either one of the following is true:

All unique key columns are null.

All unique key columns do have a value.

If legacy data causes a unique key to have mixed null and not null columns, then document this deviation in the description field of the unique key constraint.

DES-78 Unique keys are allowed to be updated, as long as they are protected by a unique index.

DES-79 For tables, document the reasons of the following deviations:

Validate in property is set to Client, Server or None.

The key constraint is not enabled.

4.5 Indexes

4.5.1 Naming Convention

4.5.1.1 Primary or Unique Key Indexes

DES-80 Designer does not create primary or unique key indexes. Oracle7 and newer versions implicitly create the index when creating primary and unique key constraints in the database. Do not create primary key indexes in the repository as this will result in problems when running the DDL scripts generated from Designer.

4.5.1.2 Foreign Key Indexes

DES-81 Name Foreign Key Indexes using the convention:

<foreign key constraint name>_I

Designer will automatically create foreign key indexes through the Database Design Transformer if it is specified in the run options.

Example:

For the foreign key FND_ACT_FK, the resulting foreign key index name would be FND_ACT_FK_I.

DES-82 In the case where multiple foreign keys were generated, and one was subsequently changed, it will be necessary to manually modify the associated index name so that it matches the renamed constraint.

Example:

Designer created:

FND_ACT_FK with index FND_ACT_FK_I

You modified the constraint to:

FND_ACT_FROM_FK

So you will need to modify the index to be:

FND_ACT_FROM_FK_I

4.5.1.3 Non-Key Indexes

DES-83 For indexes created manually that are not related to any keys, name the index using the convention:

<table alias>_<column name>_NU_I

Where 'NU' stands for non-unique.

DES-84 If the index to be created is a bit-mapped index, name the index using the convention:

<table alias>_<column name>_BM_I

Where 'BM' stands for bit-mapped.

DES-85 If the index is on multiple columns, then <column name> will be the first column in the series.

Example:

To improve query performance, an index on the column for Last Name in table Employee (EMP) is created. The index name will be:

EMP_LAST_NM_NU_I

4.5.2 Definition

DES-86 Index all foreign keys, unless you predict that the index will not deliver any performance gain or that maintaining the index will create unacceptable overhead. Document any deviations in the Description for the foreign key definition using the keyword NO FK INDEX.

DES-87 Index those columns that you reference frequently in WHERE clauses.

DES-88 The Oracle Designer tool creates indexes to match the primary, foreign and unique key constraints. This default name should not be changed.

DES-89 Do not define unique indexes, define unique key constraints instead.

DES-90 Do not deviate from the default value for Init Trans, unless you predict that the index will form a "hot spot", requiring many users to update a small number of rows simultaneously. Document any deviation from the default value in the Description belonging to the index definition using the keyword INIT TRANS.

DES-91 Do not deviate from the default value for Max Trans.

DES-92 Document any design decisions in the Description belonging to the index definition using the keyword FREE SPACE.

DES-93 In the Description for the index definition, document the reasons for any deviation between the column sequence for the index and the columns sequence in the key constraint definition. Use the keyword COLUMN SEQUENCE.

4.6 Sequences

4.6.1 Naming Convention

DES-94 Name the only sequence for a table or view using the convention:

<application_prefix>_<table/view_alias>_SEQ

DES-95 Name multiple sequences for the same table or view using the convention:

<application_prefix>_<logical_name>_SEQ

The logical name may be a column name or whatever name best defines the purpose of the sequence.

4.6.2 Definition

DES-96 Set the Code Control property to Oracle Sequence if gaps in the sequence numbering are allowed. If gaps are not allowed, you should set this property to Code Control Sequence.

DES-97 Briefly define the purpose and usage of each sequence.

DES-98 Use ascending sequences. Explicitly state any deviation from this rule in the description for the sequence definition.

DES-99 Increment sequences by a value of 1. Explicitly state any deviation from this rule in the description for the sequence definition.

DES-100 Do not cycle sequences. Explicitly state any deviation from this rule in the description for the sequence definition. Indicate how previously generated numbers are removed before the sequence “wraps”.

DES-101 Do not have sequences generated in the exact order of the request. Explicitly state any deviation from this rule in the description for the sequence definition. Indicate why the sequences should absolutely be generated in the requested order. A good exception is the use of a sequence generator as an internal “clock” to indicate the exact order in which certain events occurred by requesting a new sequence value. Minimum, Maximum

DES-102 Do not make maximum and minimum values larger than the length of the column for which the sequence is used.

4.7 Views

4.7.1 Naming Convention

DES-103 Name the view using the convention:

<table_name>_V

DES-104 The maximum length for a view name will be 26. If the name of the view exceeds 26, then use the table alias, abbreviations and acronyms as needed. View names must be singular just like tables.

4.7.2 Definition

DES-105 Define an alias for the view. This alias should be exactly three characters and should be unique across all tables and views within the application.

DES-106 Do not use a column prefix for view columns.

DES-107 Enter the display title that most likely will be used if the view is used as a module base table/view usage.

DES-108 Define the purpose and usage of the view.

DES-109 Use the same alias as used for the underlying table itself. If a table is used more than once in the view definition, add a sequence number to the table alias.

DES-110 Assign to columns in a view the same name as the columns in the underlying tables. For columns not directly based on database columns, refer to the conventions for naming columns.

DES-111 If the underlying column is in a domain, the view column should be in the same domain.

DES-112 The qualifier, if used, should give the end users a clear idea of the purpose and contents of the view. Use the *criteria* qualifier if

- Using the table name alone is not unique
- The view is based on a join of 2 or more tables
- The view contains a where clause
- The view is unusually complex.
- The view is a summary.

Examples:

CEFT_ORG_ACTIVE_V provides information on only active ORGANIZATIONS.

CEFT_ORG_VEND_V is a view joining the ORG table to the VEND table.

CEFT_ORG_BANKS_V provides information about CUSTOMERS and their NOTES of type 'BANKS'.

4.8 Database Triggers

4.8.1 Naming Convention

DES-113 Name a database trigger using the convention:

<application prefix>_T<table_alias>_<when><type>_<level>

When should be abbreviated:

- Before = B
- After = A

Types should be abbreviated:

- Insert = I
- Update = U
- Delete = D

Level should be abbreviated:

- Row = R
- Statement = S

DES-114 Extend the name of a row-level trigger, which fires on update with a logical name, if there are more than one of such update triggers.

4.8.2 Definition

DES-115 The module name should be the same as the name of the trigger.

DES-116 Define the conditions that apply to all business rules enforced in the database trigger in the Trigger When condition.

4.9 Application Design Transformer

There are no applicable standards for the Application Design Transformer.

4.10 Modules

4.10.1 Scope

The names of all modules developed with and generated from Oracle Designer will conform to these conventions. This includes all menus, reports, screens, libraries, and webserver modules. In addition, all SQL*Plus reports (where documented) developed at DCII will also conform to these conventions. The Director of DCII Engineering must approve changes to these standards.

DES-117 The following distinctions between modules developed for or associated with Legacy systems and Vendor-supplied software and those to be developed for new or replacement systems are recognized:

- All new or replacement systems developed for DCII will be developed using the DPET-approved release of the Oracle Designer tool set. All modules, whether custom designed or generated from Designer, will adhere to standards without exception.
- All modules that use a Designer supported language must be stored in the repository.
- Legacy modules' names must be changed to conform to the DFAS standard.
- Vendor's modules' names will not be changed. If objects from a commercial-off-the-shelf Package (COTS) are recovered into the Designer repository, and any modules are generated in-house to use those vendor objects, the new modules will be developed according to this standard.

4.10.2 Naming Convention

4.10.2.1 Modules

DES-118 Name Module Short Names for primary modules using the convention:

<application prefix>_<module type identifier><descriptor>

where <descriptor> is a name meaningful to the development team responsible for the module.

See Appendix B for approved application prefixes.

Module type identifiers are:

- F = Form
- K = Package (PL/SQL)
- L = Developer Library
- M = Menu
- O = Object Library
- P = Procedure (PL/SQL)
- R = Report
- S = Shell
- T = Triggers (PL/SQL)
- U = Function (PL/SQL)
- W = Web Form
- X = Template Form

Since DCD utility packages are available for all applications and teams to use, they are themselves standard and their names shall be DCD_<descriptor> where <descriptor> is a meaningful word indicating the purpose of the utility.

The DSDS application also uses utility packages specific to DSDS. They are standard and their names shall be DS_<descriptor> where <descriptor> is a meaningful word indicating the purpose of the utility within the DSDS application.

DES-119 Module Short Names should not exceed 20 characters.

DES-120 Define Module Names using logical names, without special characters.

Screen Example:

Short Name: FICS_FUPDATE

Translation: File Inventory Control Subsystem, Form, Update Function

Reports Example:

Short Name: FICS_RUPDATE

Translation: File Inventory Control System, Report, Updates

4.10.2.2 Module Components

All Module Components (MC) must include the alias of the base table name upon which the module is based.

Example: INV is the MC name for a component based on the Invoices table

DES-121 If there will be multiple components within a module using the same base table, then the name will include an underscore followed by an abbreviation of the purpose.

Example: INV_QRY is the MC name for a second component based on the Invoices table that will be query only

4.10.2.3 Module Component Elements

4.10.2.3.1 Item Groups

DES-122 Item groups may be named to represent a functional grouping of data. The name may be multiple words with no underscores between them.

Example: The item group encompasses columns that make up the information for a mailing address so the item group is named MAILING ADDRESS

DES-123 If the layout item group is a horizontal item group, the name will be prefixed with an "H". If it is a vertical item group, it will be prefixed with a "V". (Additional prefixes may be developed to represent the additional functions for which item groups are used in reports generation.)

Example: H MAILING ADDRESS is a horizontal item group containing columns for a mailing address

DES-124 If the purpose of the item group is to enable the generation of a specific layout then the name will be LAYOUT #, where # represents an integer.

Example: V LAYOUT 1 is a vertical item group used specifically for layout generation

DES-125 If nested item groups are used to achieve a complex layout then the name of the nested item groups will be NESTED LAYOUT #-#, where the first # represents the number associated with the parent layout group and the second # is a sequential integer within that group

Example: H NESTED LAYOUT 1-2 represents the 2nd nested item group within the item group named LAYOUT 1. It is a horizontal item group.

Example: V NESTED LAYOUT 1-2-1 represents the 1st nested item group within the 2nd nested item group within the item group named LAYOUT 1. It is a vertical item group.

4.10.2.3.2 Unbound Items

DES-126 All unbound items will be prefixed with UB and an underscore followed by text that describes its function. Approved abbreviations and acronyms may be used as necessary.

Example: UB_TOTAL_PRICE is an unbound item that will contain the results of a calculation for total price

4.10.2.3.3 SQL Query Sets

DES-127 All SQL Query sets will be named with the word UNION and the table alias of the base table usage in the module component.

Example: UNION EMP is the name of the query set containing a reference to the EMPLOYEES base table usage

4.10.2.3.4 Navigation Action Items (Buttons)

DES-128 Navigation action items will be named using a prefix of NA followed by the module component name followed by the target module component or module name with an underscore in between

Example: NA_CUST_FNAPF010 indicates a button to navigate from the CUST component in the current module to the module FNAPF010

4.10.2.3.5 Custom Action Items (Buttons)

DES-129 Custom action items will be named using the prefix CA followed by text that describes the function of the action item. Approved abbreviations and acronyms may be used as necessary.

Example: CA_CALC_TOTAL indicates a button that when pressed will cause a total to be calculated

4.10.2.3.6 Application Logic Event Code Segments

DES-130 All code segments entered to implement custom application logic will be named using a text string that defines the purpose of the code. Approved abbreviations and acronyms may be used as necessary.

Example: Use parameter value when present

4.10.2.3.7 Application Logic Named Routines

DES-131 Named routines will conform to the same standard set forth for PL/SQL procedures

4.10.2.3.8 API Logic Code Segments

DES-132 API Logic code segments will conform to the same standard set forth for Application Logic code segments.

4.10.2.4 Named Preference Sets

DES-133 All preference sets will be defined by the DCII Common Service Functions group. Name preference sets using the following convention:

CSF_<descriptor>

Where the descriptor is a brief explanation of the purpose of the preference set.

4.11 PL/SQL, SQL, CTL, PAT, AWK, SHL and KSH

4.11.1 Naming Conventions

See 4.10.2.1 Module Naming Convention

4.11.2 General Standards

DES-134 PL/SQL modules are considered database objects and are generated via the Server Generator, not the client Generators used by screens and reports. Placing packages, procedures, functions, and database triggers in the server tends to reduce traffic across the network.

DES-135 PL/SQL provides a mechanism to manipulate data procedurally. Thus you can use SQL statements to manipulate data while using flow control statements to process this data. PL/SQL is a block-structured language. The basic units that make up a module (packages, procedures, functions, and anonymous blocks) are logical blocks, which in turn can contain other nested blocks.

A PL/SQL block has three sections:

- Declarative section
- Executable section
- Exception-handling section

DES-136 All features of the PL/SQL language are allowed unless the feature has been shown to be unsafe. The GOTO statement is one such feature that has been shown to be unsafe, and will not be used.

DES-137 All PL/SQL modules must be written in such a fashion that an experienced PL/SQL programmer can maintain the software without undue recourse to other documents. All modules must be well documented. At a minimum this includes the purpose of the module, a point of contact, and a history section.

DES-138 The preferred method for storing modules in the Designer repository is the free-format method.

DES-139 All PL/SQL procedures and functions will be implemented as part of a PL/SQL package. There will be no standalone PL/SQL procedures or functions implemented as part of the production system.

4.11.3 Documentation and Formatting

DES-140 Inline documentation should be used to clarify and or document individual parts of a SQL statement, and temporarily disable part of a SQL statement.

DES-141 Any change made to an object (e.g. enable or disable index use or hints used for the cost-based optimizer) should include comments documenting the changes as well as the original condition of the statement.

For TDRs, fill in the required information PLUS copy the TDR subject line into the "Change Made (brief description) area.

A Change History Comment shall be added to ALL scripts and will contain the following information:

<u>Date</u>	<u>SCR# (or TDR#)</u>	<u>Rel#</u>	<u>Developer</u> <u>(last name, first initial)</u>	<u>Change Made</u> <u>(brief description)</u>
-------------	-----------------------	-------------	---	--

Table format (ABOVE) OR in Linear format (BELOW)

Date
SCR# (or TDR#)
Rel#
Developer (last name, first initial)
Change Made (brief description)

DES-142 Simple comments shall be documented for CTL, PAT, AWK, SHL, KSH, SQL and PL/SQL objects as follows:

-1- CTL or PAT files shall be commented by putting a "--" in the first 2 positions of the 1st line.

CTL or PAT Example:

```
-- Date      SCR# (or TDR#)  Rel#  Developer      Change Made
--                                     (last name, first initial)  (brief description)
load data
into TABLE nsa_ay_detail
append
(
  nh_batch_file_id      constant '&batch_num',
  rec_seq_id            sequence (1, 1),
```

-2- AWK files shall be commented by putting a "#" in the first position of the 2nd line.

AWK Example:

```
#!/usr/bin/awk -f

# Date      SCR# (or TDR#)  Rel#  Developer      Change Made
#                                     (last name, first initial)  (brief description)

BEGIN{
  htot = 0
  dtot = 0
  tot = 0
  s1=0
  s2=0
}
```

-3- SHL and KSH files shall be commented by putting a # in the first position on the 1st line

SHL and KSH Examples:

```
# -----
# Name      : nsa_somards_main.shl
#
# Description : SOMARDS main routine. Handles call of every sub-
#               routine
#               needed to format, validate and load a SOMARDS file.
#
# Returns    : the batch id assigned for the file being processed.
```

```

#
# Parameters : Seq Name      Description
#             1 dir_filename Directory and Filename.
#
# Usage      : nsa_somards_main.shl <dir_filename>
#
# Called by   : nsa_somards_load.shl
# Calls       : nsa_chkamt.awk
#              nsa_somards_dtetime.awk
#              nsa_somards_fmt.awk
#              nsa_fmtatt.awk
#              nsa_header_upd.sql
#              nsa_detail_del.sql
#              ay_post.sql
#              nsa_k_utl.timediff
#
# Revision History:
# Date      SCR# (or TDR#)  Rel#  Developer      Change Made
#              (last name, first initial)  (brief description)
# 01-17-2001                Sormillon, J    Revised Source
#                               System/Site
#                               validation to
#                               take site id
#                               from header
#                               rather than from
#                               the filename
# -----

```

-4- SQL and PL/SQL files shall be commented by putting "--" before the comment or using /* and */ around the comments

For a given PL/SQL package:

Change information to the SPEC shall be documented in the SPEC using the format below.

Change information to the PACKAGE BODY shall be documented in the PACKAGE BODY using the format below.

Also, the change information shall be documented / stored in Designer in the NOTE property using the format below.

```

-- Date      SCR# (or TDR#)  Rel#  Developer      Change Made
--              (last name, first initial)  (brief description)

```

Table format (ABOVE) OR in Linear format (BELOW)

```

-- Date
-- SCR# (or TDR#)
-- Rel#
-- Developer (last name, first initial)
-- Change Made (brief description)

```

SQL and PL/SQL Examples:**Single Line Example:**

```
-- statement created on 05/10/00 by Ron Plew.
select t1.column1, t1.column2, t1.column3, t1.column4,
       t1.column5, t1.column6
from table1 t1;
```

Comment within a SQL statement:

```
select t1.column1, t1.column2,
       t1.column3, t1.column4,
       t1.column5, t1.column6 -- added column6 10 May 00, Ron Plew
from table1 t1;
```

Multi-line comment brackets (/* & */) should be on lines by themselves in columns 1 and 2.

Example:

```
/*
This statement was written on 05/10/00 by Ron Plew.
The purpose of the statement is to return specific data quickly
and efficiently.
*/
select t1.column1, t1.column2, t1.column3, t1.column4,
       t1.column5, t1.column6
from table1 t1;
```

Combination Example:

```
-- Select all Conflicting Job Responsibilities
select jr_nm2 -- get job responsibility in right column where entered JR matches JR in left column
from csf_cnflct_of_int
where jr_nm = :acs.jr_nm
union
select jr_nm -- get job responsibility in left column where entered JR matches JR in right column
from csf_cnflct_of_int
where jr_nm2 = :acs.jr_nm;
```

DES-143 A standard header must be used for all PL/SQL modules. It must be placed between the name and the "IS".

Example:

```
PACKAGE package_name IS
/*
|| Author: S. Feuerstein 11/95
||
|| Overview: Manage list of selected items correlated with a block on the
|| screen
||
|| Major modifications (when, who, what)
|| 12/94 - SEF - Create package
|| 3/95 - JRC - Enhance to support coordinated blocks
*/
```

DES-144 A header must be used for the bodies of all modules that are database objects (functions, procedures, package bodies, and triggers). The modification information should include changes in the implementation due to maintenance, modifications etc.

- DES-145 Use consistent spacing for indentation of all lexical elements. Each new lexical level should be further indented. Three spaces is recommended for indentation.
- DES-146 Align parameter passing modes. The first parameter in the list can either be on the same line as the subprogram name or on the line following.

Example:

```

PROCEDURE sample (copied_in      IN      some_type,
                  copied_in_out  IN OUT  some_other_type,
                  copied_out     OUT    yet_another_type);
or-
PROCEDURE sample (copied_in      IN      some_type,
                  copied_in_out  IN OUT  some_other_type,
                  copied_out     OUT    yet_another_type);

```

- DES-147 Vertically align major block-keywords. As an option, the keyword 'IS' may be included in this alignment.

Example # 1:

```

DECLARE
...
BEGIN
...
EXCEPTION
...
END <name>;

```

Example # 2:

```

PROCEDURE ... IS
...
BEGIN
...
EXCEPTION
...
END <name>;

```

Example # 3:

```

FUNCTION ... RETURN...
IS
...
BEGIN
...
EXCEPTION
...
END <name>;

```

4.11.4 Data Load Standards

- DES-148 Whenever possible, the relationship from staging table to target table should be 1:1.
- DES-149 All data loads shall capture throughput data. This can be accomplished via PL/SQL calls to the CSF_TIMING package.
- DES-150 All data loads shall provide debugging messages via the PL/SQL package CSF_DEBUG.
- DES-151 All data loads will have periodic commits. The commit rate will be parameterized to facilitate tuning.

- DES-152 All staging table data will be deleted immediately after successful processing (i.e. record at a time). Data that contains errors is to remain in the staging table.
- DES-153 Error handling and messages should be meaningful and provide sufficient information to fix errors. For example, an additional column (ERR_TXT VARCHAR2(2000)) in a staging table provides a good method for writing error information for a single record. CSF_ERROR_LOG provides a good method for global messages relative to a load.
- DES-154 Grouping all PL/SQL procedures into a single package is preferred over standalone procedures. For example, a Navy load package might contain all of the procedures for loading Navy data.

4.11.5 PL/SQL Coding Standards

- DES-155 Use underscores to separate words within an identifier.
- DES-156 Make PL/SQL keywords distinguishable from other elements of the program. There are at least two common standards in existence.
- Keywords should be in uppercase and user-defined identifiers in lowercase.
 - Keywords should be in lowercase and user-defined identifiers in Initcaps.
- Neither addresses Built-in function and package names (LENGTH, DBMS_OUTPUT etc.)
- DES-157 Spell out identifiers completely unless there is a common, unambiguous abbreviation that takes up significantly fewer characters. Use the standard DFAS abbreviations
- DES-158 Avoid using database column names or database table names as the names of variables.
- DES-159 Variables declared at the outermost level of a package body should be prefixed with 'g_'.
- DES-160 Boolean variables should be named to indicate a true/false proposition. Variables of all other datatypes should be nouns.

Example:

```
Account_is_open      Boolean;
Account_Name         VARCHAR2 (100);
```

- DES-161 Name a parameter in terms of its mode (IN, OUT, IN OUT). Use the mode as a suffix. Examples:

```
PROCEDURE Place_call
(company_id_in      IN      NUMBER,
call_type_in_out   IN OUT VARCHAR2,
company_rm_out     OUT VARCHAR2)
```

- DES-162 Name a procedure to describe the action taken (verb-noun structure, such as 'calculate_totals').
- DES-163 The rules for function names should be the same as the rules for variables (see NAM-002). That is, the name of a Boolean function should obey the same rules as the name of a Boolean variable.
- DES-164 The name of a cursor should be descriptive and should end with the suffix '_cur'. The name of a record anchored to a cursor or the name of a control variable in a cursor FOR-loop should be the same name as the cursor but should use the suffix '_rec' instead of '_cur' (See NAM-014).
- DES-165 The name of a user-defined type should have a suffix '_type'

Example # 1:

```
TYPE strings_type IS TABLE OF VARCHAR2(100)
INDEX BY BINARY_INTEGER;
```

Example # 2:

```
my_list_tab strings_type;
```

- DES-166 Record variables should be suffixed with “_rec”. This should be true irrespective of the category of record (user-defined, table-based, and cursor-based).

Example:

```
TYPE name_type IS RECORD
(
    first_name VARCHAR2(20),
    last_name  VARCHAR2(20)
);
```

```
person_rec name_type;
```

- DES-167 PL/SQL collection (index-by, VARRAY and nested tables) variables should be named to indicate their collective nature. This can be accomplished by naming the collection as a plural noun or by distinguishing the variable name with a collective suffix such as ‘_tab’, ‘_tbl’ or ‘_list’.
- DES-168 Name each block statement (anonymous block) with a block label; include that same label between the END of the module and the semicolon.
- DES-169 Include the name of any module (package spec, package body, function, procedure, block statement) between the END of the module and the semicolon.
- DES-170 For a numeric FOR loop, incorporate the word “index” or “counter” or something similar into the name of the loop index.

Name each loop with a loop label and repeat the label following the END LOOP. This standard can be ignored in the case of very small loops whose intent is easily discerned.

Example:

```
FOR year_index IN 1..12 . . .
...
...
END LOOP; -- year_index
```

4.11.5.1 PL/SQL Arguments

- DES-171 Name Arguments for PL/SQL packages and procedures using the following convention:

p_<argument name>

Where <argument name> is a logical, meaningful, and concise name representing the value that will be passed to the program unit. Where the argument will be used to hold the value of a column in a SQL statement, then <argument name> must be the same as the column name.

- DES-172 Precede the <argument name> with “P” so that there are no conflicts with database object names or confusion as to which are PL/SQL arguments and which are database objects. All arguments must be in lower case, excluding the prefix.

Example: p_header_id

4.11.5.2 PL/SQL Variables

DES-173 Name PL/SQL program variables using the following convention:

v_<variable name>

Where <variable name> should be a logical, meaningful, and concise name representing the value the variable will hold. Where the variable will be used to hold the value of a column in a SQL statement, then <variable name> must be the same as the column name.

DES-174 Precede the <variable name> with “v” so that there are no conflicts with database object names or confusion as to which are PL/SQL variables and which are database objects. All variables must be in lower case, including the prefix.

Example: v_header_id

4.11.5.3 PL/SQL Constants

DES-175 Name PL/SQL program constants using the following convention:

c_<constant name>

Where <constant name> should be a logical, meaningful, and concise name representing the use of the value the constant will hold. Where the constant will be used to hold a value to be assigned to a column in a SQL statement, then <constant name> must be the same as the column name.

DES-176 Precede the <constant name> with “c” so that there are no conflicts with database object names or confusion as to which are PL/SQL constants and which are database objects or variables. All constants must be in lower case, including the prefix.

Example: c_header_id

4.11.5.4 PL/SQL Global Variables (forms only)

DES-177 Name Oracle Forms global variables using the following convention:

global.<variable name>

Where <variable name> should be a logical, meaningful, and concise name representing the value the variable will hold. Where the variable will be used to hold the value of a field in the data block, then <variable name> must be the same as the field name.

Global variables must always be referenced with ‘:global.’ preceding the variable name. Therefore, no additional prefix is required to distinguish global variables from local or standard variables. In order to avoid confusion, do not use the same name for both a global and a local variable.

Example: header_id (i.e., :global.header_id)

4.11.5.5 PL/SQL Declarative Section

DES-178 Each declaration should begin on a new line.

DES-179 Whenever possible, anchor variables, record components, PL/SQL table components, and parameters to the appropriate database tables, database columns and cursors.

NOTE: When providing packaged resources to users, in some circumstances, anchoring has a downside. Consider the situation where the user has access to the package but does not have access to the underlying database tables. In this case, anchoring does not give the information needed in order to use the spec. It might be preferable to describe the interface (records, PL/SQL tables, subprogram parameters etc.) in terms of standard datatypes instead of anchored datatypes.

- DES-180 Declare variables as constants if their values do not change throughout the code.
- DES-181 Specify a full column list (as opposed to using ‘*’) in each cursor declaration unless all the columns are selected AND input into a variable declared with the <cursor>%ROWTYPE (as opposed to inputting into a list of variables). In the case of tables with a large number of columns, the programmer may opt to use SELECT * if the majority of columns are being used in the SELECT statement.
- DES-182 Explicitly declare exceptions for each Oracle error condition that is expected in the normal execution of the program and is not already mapped to an Oracle exception name. Use the pragma, EXCEPTION_INIT, to map the exception to the Oracle error number.
- DES-183 Provide well-delimited sections for the following categories: Cursor declarations, type and subtype declarations, variable and constant declarations, exception declarations, and subprogram declarations. The variable declaration section may be further decomposed into scalar variables, record variables and collection variables.
- DES-184 To avoid hardcoding of literals, declare named constants for use in the executable portion of a block.

4.11.5.6 PL/SQL Executable Section

- DES-185 Do not use GOTO statements.
- DES-186 Close all explicitly opened cursors. This is especially important for cursors declared in packages. Cursors declared in other blocks are implicitly closed when the block is abandoned; close them anyway.
- DES-187 Use explicit datatype conversion functions instead of implicit conversions. An exception is the concatenation of items in a call to DBMS_OUTPUT.PUT_LINE for debug purposes.
- DES-188 Remove all hard coded values (except for 0 and 1) for the program and replace them with named constants (see DEC-012) or functions defined in packages. It is easier to change a named constant once in a separate area than to catch and change each hard coded value in a program or function.
- DES-189 Never exit from a numeric or cursor FOR loop or a WHILE loop with an EXIT or RETURN.
- DES-190 Always test for NULL with ‘<expression> IS [NOT] NULL’.
- Remember that NULL is never equal to any value and is never not equal to any value.
- DES-191 In a package that calls the built-in functions, USER and SYSDATE, add the following variables and refer to them instead of the functions:

```
g_user      VARCHAR2(50) := USER;
g_sysdate   DATE         := SYSDATE;
```

4.11.5.7 PL/SQL Exception Handling

- DES-192 Trap predefined oracle exceptions by name. Do not use WHEN OTHERS and then discriminate on the SQLCODE function result.

- DES-193 All expected Oracle error codes should be handled with the predefined Oracle exception name or a user defined exception name that has been mapped with pragma EXCEPTION_INIT procedure.
- DES-194 Do not use exceptions to perform branching logic.
- DES-195 Each exception handler for a function must return a value or raise an exception (either the same one using RAISE or a different using its name). Remember that if a function reaches its END, an exception (function returned without value) is raised. The compiler will check to see that your function contains a RETURN but will not check to see that your EXCEPTION handler part has a RETURN.
- DES-196 When using the WHEN OTHERS clause, capture (display, log, etc.) the actual error condition (using the built-in function, SQLERRM) in the exception handler.

4.11.6 SQL Coding Standards

4.11.6.1 General Layout of SQL Statements

- DES-197 Lowercase should be used when writing SQL statements.

Note: Literal constants may require uppercase characters.

- DES-198 Start each clause that contains a column name, table name, or SQL reserved word on a new line.

- DES-199 Align code by using spaces to indent. Do not use tabs.

Example:

```
select  t1.column1, t1.column2
        t1.column3, t1.column4
from table1 t1
where exists (select t2.column1
              from table2 t2)
and t1.column1 > value1
order by 1;
```

4.11.6.2 SQL Statements - General

- DES-200 Put single space after every keyword, identifier, comma, and operator. Do not put a space between scalar or set functions, or within parenthesis. An exception for the parenthesis is: When enclosed in parenthesis, a comma should be followed by a space.

Example:

```
select t1.column1, t1.column2, t1.column3 * value1,
       min(t1.column4), t1.column5, t1.column6,
       substr(t1.column7, 1, 3)
from table1 t1
where t1.column1 > value2;
```

- DES-201 Use aliases for tables and columns. The table alias should be the initials from the table name or an abbreviation. The alias should be less than ten characters. Similar names will be suffixed with numbers. Prefix each column with the table alias followed by a period and then the column name. Column names should denote any conversions of data and source table.

Example:

```
select t1.column1, t1.column2, addr.column3,
       addr.column4, mt.column5, mt.column6,
```

```

        to_char(mt.column1) mt_column1_char
from table1 t1,
   address addr,
   my_table mt;

```

- DES-202 Use the *NVL* function on any column that can contain null values when it is being compared to an actual value or when it is being used in a calculation.

Example:

```

select nvl(t1.column1, 0) * value1
from table1 t1
where nvl(t1.column5, 0) <= value2;

```

4.11.6.3 SQL Statements – Select

- DES-203 Start the *select* clause on a separate line. If a long column list requires another line, indent the line.

Example:

```

select column1, column2, column3, column4,
       column5, column6

```

- DES-204 Use spaces to follow the *select* keyword and columns except for the last column in the list.

Example:

```

select column1, column2, column3, column4

```

- DES-205 Do not use the asterisk (*) to select all columns. Specifically list the columns required.

- DES-206 When using expressions, make sure the expression is the last operation performed, especially in *group by* functions. For example, multiply the result of a *MAX* function by 1.5, instead of each value, to determine the maximum value.

Example:

```

select max(t1.number1) * value1 max_val
from table1 t1;

```

- DES-207 To use a hint in a *select* statement, the hint must immediately follow the keyword *select* separated by a single space.

Example:

```

select /*+ full(table1) */

```

4.11.6.4 SQL Statements – Union, Intersect, and Minus

- DES-208 The *union*, *intersect*, and *minus* should be on a line of their own.

Example:

```

select column1, column2
from table1
union
select column1, column2
from table2;

```

4.11.6.5 SQL Statements – From

- DES-209 Start the *from* clause on a separate line. Put only one table per line followed by a comma except after the last table in the list. A single space must be between the table name and the alias.

Example:

```

from table1 t1,
    table2 t2,
    table3 t3

```

4.11.6.6 SQL Statements – Where

DES-210 Start the *where* clause on its own line. Put each condition on a separate line. The most restrictive condition should go last in the *where* clause.

Note: notice the *and* was indented.

Example:

```

where column1 > 0      -- least restrictive condition
    and column2 = '1'
    and column3 >= '2' -- most restrictive condition

```

DES-211 Place join conditions at the beginning of the *where* clause.

Example:

```

Where column1 = column1 -- join
    and column2 = column2 -- join
    and column2 = '1'    -- condition

```

DES-212 Use the *IN* operator instead of multiple *OR* conditions on the same column.

Example:

```

where column2 in ('1', '3', '5', '7')

```

Instead of

```

where column2 = '1'
    or column2 = '3'
    or column2 = '5'
    or column2 = '7'

```

DES-213 When using the *OR* condition with different columns, use parentheses around the *OR* conditions.

Example:

```

where (column2 = '1'
    or column3 = '3')

```

4.11.6.7 SQL Statements – Group By and Having

DES-214 Only group on the columns used by the *select* clause.

Example:

```

select t1.column1, t1.column4, count(*), max(t1.column5)
from table1 t1
where t1.column3 = value1
group by t1.column1, t1.column4;

```

DES-215 Use as many conditions in the *where* clause as possible rather than in the *having* clause. This prevents the *having* clause from placing the conditions since indexes are not used by the *having* clause.

DES-216 The *having* clause should only contain group functions for limiting the data.

Example:

```
select t1.column1, t1.column4, count(*), max(t1.column5)
from table1 t1
where t1.column3 = value1
group by t1.column1, t1.column4
having count(*) > value2;
```

4.11.6.8 SQL Statements – Connect By

DES-217 Always specify a starting point using the *start with* clause. This provides clarity for the usage of the *connect by* clause.

DES-218 Do not use the *connect by* clause with recursive data.

4.11.6.9 SQL Statements – Order by

DES-219 Use the *order by* clause only if there is a requirement to sort the data.

DES-220 Only use numbers for the *order by* columns when the set operators (*union*, *minus* and *intersect*) are being used. Otherwise, use column names.

Example:

```
select t1.column4, t1.column5, t1.column8
from table1 t1
order by t1.column4, t1.column5;
```

Example using set operators

```
select t1.column4    emp_name,
       t1.column5    emp_dept,
       t1.column8    emp_loc
from table1 t1
union
select t2.column1,
       t2.column2,
       t2.column10
from table2 t2
order by 1, 2, 3;
```

DES-221 If a mix of ascending and descending order sorts is used, indicate the type of sort on all *order by* columns.

Example:

```
select t1.column4, t1.column5, t1.column8
from table1 t1
order by t1.column4 desc,
       t1.column5 asc,
       t1.column8 desc;
```

4.11.6.10 SQL Statements – For Update

DES-222 The column selected in the *for update* clause will be the primary key column of the table being updated.

Example:

```
select t1.column4, t2.column1, t2.column3
from   table1 t1,
       table2 t2                -- primary table
where  t1.column1 = t2.column1
for update of t2.column1;      -- primary key
```

- DES-223 The NOWAIT option of the *for update* clause will not be used in SQL. It will be used in PL/SQL only, where the exception can be handled.

4.11.6.11 SQL Statements – Insert

- DES-224 The keyword *values* should be on a separate line.

Note: Notice indentation of column listing and values.

Example:

```
Insert into table1 (column1, column2, column3,
                  column4, column5)
values
(value1, value2, value3, value4, value5);
```

- DES-225 Use a full column list for an *insert* statement for the target table and for the *select* list when it exists.

Example:

```
insert into table4 (column1, column2, column3)
select t2.column3, t2.column4, t2.column5
from table2 t2;
```

- DES-226 When no value is being inserted into a column, specify the *NULL* value. Do not leave the column off the column list.

Example:

```
insert into table4 (column1, column2, column3)
select t2.column3, null, t2.column5
from table2 t2;
```

4.11.6.12 SQL Statements – Update

- DES-227 List each column being updated on a separate line.

Note: Although the *where* clause is optional, the use of the *where* clause is highly recommended. If not used, all rows will be updated with the new value.

Example:

```
update table1 t1
set t1.column1 = value,
    t1.column2 = value
where t1.column1 = value;
```

- DES-228 The ROWID column can only be used in an *update* when a *select for update* statement has locked the rows.

4.11.6.13 SQL Statements – Delete

- DES-229 The format of a delete statement has delete on the first line and from table name on the second line. This is to emphasize that a delete is taking place.

Note: Although the *where* clause is optional, the use of the *where* clause is highly recommended. If not used, all rows will be deleted.

Example:

```
delete
```

```
from table1 t1
where t1.column1 = value;
```

4.11.6.14 SQL Statements – Subqueries

DES-230 The column list in the *where* clause of the main query should not contain concatenation or conversion functions. All conversions or concatenation should be done in the *select* list of the subquery.

Example:

Note indentation

```
select t1.column1, t1.column2
       t1.column3, t1.column4
from table1 t1
where exists (select 'x'           --subselect
              from table2 t2
              where t2.column1 = t1.column2)
and t1.column1 > value1
order by t1.column1;
```